

Systèmes et applications asynchrones
Middleware à message

Roland Balter
ScalAgent Distributed Technologies

Roland.Balter@scalagent.com

www.scalagent.com

Plan

- Caractérisation des systèmes asynchrones
- Modèles pour la programmation asynchrone
- Middleware asynchrones (MOM)
- JMS : un exemle de pprogrammation asynchrone
- JORAM : un exemple de MOM

© ScalAgent Distributed Technologies – 2001-2003

Architecture distribuée

- Application/système distribué
  - Ensemble de composants logiciels coopérants
  - Coopération = communication + synchronisation
- Eléments de choix d'une architecture distribuée

API	**CORBA, JMS, . .**
Modèle de programmation communication & synchronisation	**Synchrone :Client-serveur (RPC, RMI, ORB, etc.)** **Asynchrone : messages et événements** **Objets mobiles** **Objets partagés**
Middleware services systèmes pour gestion de ressources distribuées	**Flots d'exécution, mémoire, persistance** **Fiabilité, disponibilité, sécurité, scalabilité, . .**
Système d'exploitation + TCP/IP	

© ScalAgent Distributed Technologies – 2001-2003

Caractéristiques du mode "asynchrone"

- Systèmes faiblement couplés
  - Couplage temporel : systèmes autonomes communicants
    - *Communication « spontanée » en mode « push »*
    - *Fonctionnement en mode déconnecté : site absent ou utilisateur mobile*
  - Couplage spatial : systèmes à grande échelle
    - *Fonctionnement en mode partitionné : pannes temporaires de réseau*
    - *Communication « anonyme » : non connaissance des correspondants*
- Simplicité
  - Modèle de communication « canonique »
    - *Envoi de message*
    - *Base universelle : TCP-UDP/IP*
    - *Gestion de l'hétérogénéité : systèmes, réseaux*

© ScalAgent Distributed Technologies – 2001-2003

Un peu d'histoire

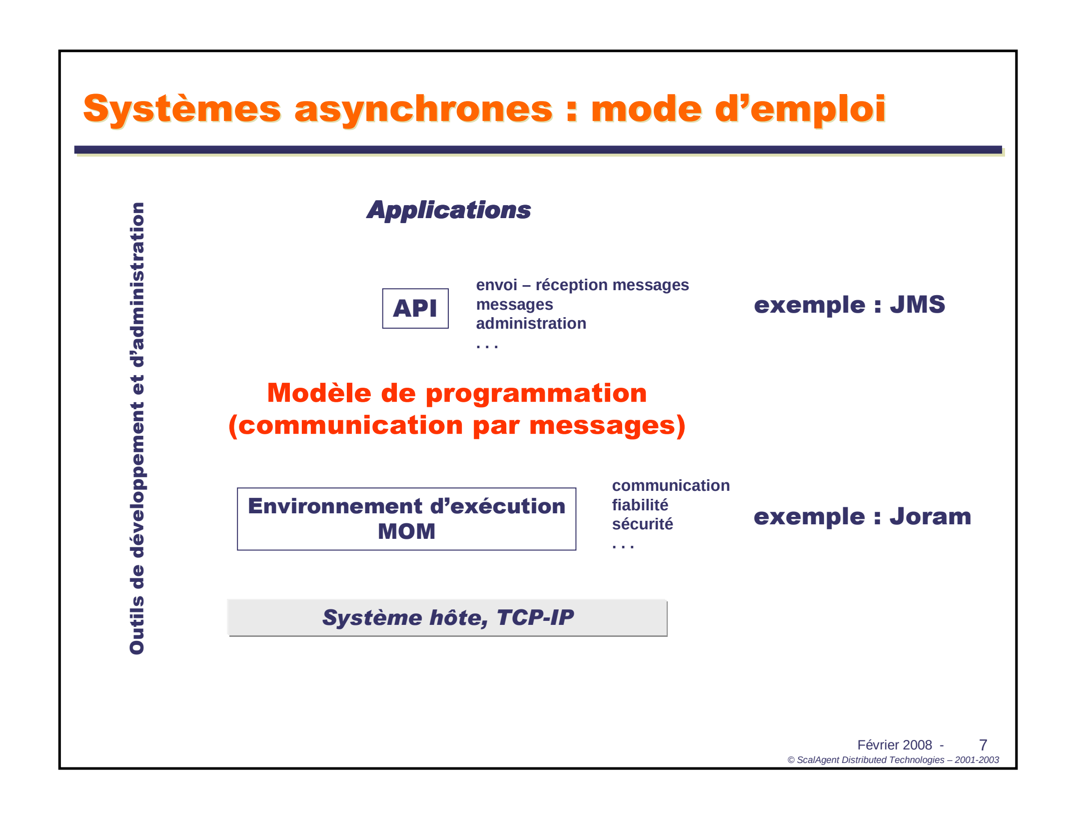
- Les systèmes asynchrones sont en usage dans le monde de l'Internet depuis longtemps
  - Le courrier électronique (communication point-à-point)
    - *le producteur envoie un message à un destinataire qu'il connaît*
    - *le message est stocké sur un serveur, le consommateur reçoit ultérieurement le message lorsqu'il se connecte*
  - Les listes de diffusion (communication multi-points)
    - *Le message est diffusé à tous les éléments de la liste*
  - Les news (Anonymat, Publish/Subscribe)
    - *le consommateur s'abonne à une liste de diffusion*
    - *le producteur publie une information dans un forum*
    - *le consommateur lit le contenu du forum quand il le souhaite*

© ScalAgent Distributed Technologies – 2001-2003

Les usages des systèmes asynchrones

- **Supervision**
  - Parc d'équipements distribués
  - Applications distribuées
- **Echange et partage de données**
  - Envoi de documents (EDI)
  - Mise à jour d'un espace de données partagées distribuées
- **Intégration de données**
  - Alimentation d'un datawarehouse/datamart depuis des sources de données hétérogènes autonomes : ETL (Extract – Transfer – Load)
- **Intégration d'application**
  - Intra-entreprise : EAI (communication, routage, workflow)
  - Inter-entreprises : B2B et Web Services (communication, orchestration)
- **Informatique mobile**
  - Communication entre équipements mobiles (souvent déconnectés) et serveurs d'appplication

© ScalAgent Distributed Technologies – 2001-2003



Communication par messages : modèle de programmation

Mode de synchronisation

- *Communication asynchrone*
  - *émission non bloquante*
  - *réception bloquante (attente jusqu'à arrivée d'un message, ou retour d'erreur)*

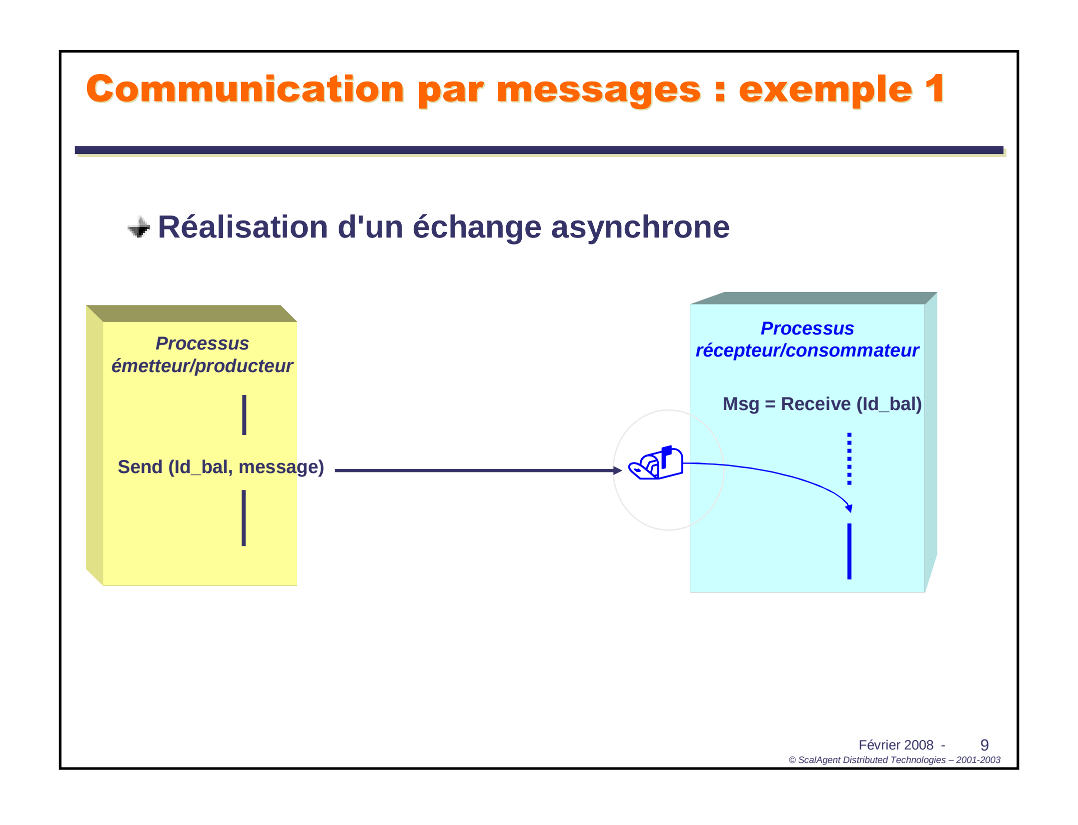
Mode de communication (désignation)

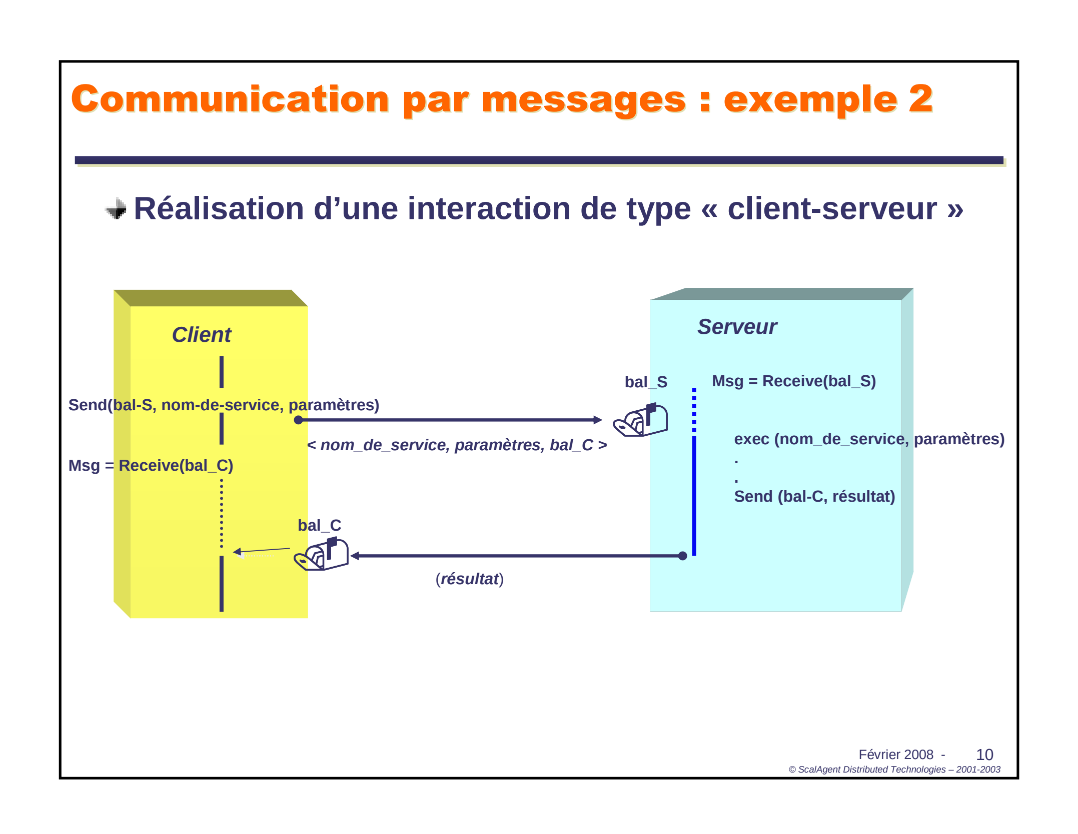
- communication directe entre processus (*acteurs, agents, . . .*)
- communication indirecte entre processus via des objets intermédiaires (*portes, boîtes aux lettres, queues de message, etc.*)

Mode de transmission

- messages possiblement typés

© ScalAgent Distributed Technologies – 2001-2003


Communication par messages : exemple 1
Réalisation d'un échange asynchrone
Processus émetteur/producteur
Send (Id_bal, message)
Processus récepteur/consommateur
Msg = Receive (Id_bal)
Février 2008 - 9
© ScalAgent Distributed Technologies – 2001-2003


Communication par messages : exemple 2
Réalisation d’une interaction de type « client-serveur »
Client
Send(bal-S, nom-de-service, paramètres)
Msg = Receive(bal_C)
< nom_de_service, paramètres, bal_C >
bal_S
Serveur
Msg = Receive(bal_S)
exec (nom_de_service, paramètres)
Send (bal-C, résultat)
bal_C
(résultat)
Février 2008 - 10
© ScalAgent Distributed Technologies – 2001-2003

Communication par messages : exemples d’environnements existants

- **Environnement de type "micro-noyau"**
  - mécanisme et primitives de base
  - exemples : Chorus, Mach/OSF-1
- **Environnement "à la unix"**
  - "sockets"
- **Environnement de programmation parallèle**
  - PVM et/ou MPI
- **Environnement d'intégration d'applications (et/ou de données)**
  - middleware à messages: MOM
  - interface de programmation ad hoc
    - *tentative de normalisation via Java JMS*

© ScalAgent Distributed Technologies – 2001-2003

Les éléments de définition d’un système de messagerie

- **Modèle**
  - Structure des messages
  - Mode de production des messages : non bloquant
  - Mode de désignation : indirect, groupe, anonyme
  - Mode de communication : point à point, multi-points
  - Modesde consommation des messages : push, pull
- **Environnement d’exécution (*run time*)**
  - Architecture : centralisée, partitionnée, répartie
  - Qualité de service : fiabilité, sécurité, scalabilité, etc.
- **API**
  - JMS

© ScalAgent Distributed Technologies – 2001-2003

Format des messages

- **Entête**
  - Information permettant l'identification et l'acheminement du message
    - *Id. unique, destination, priorité, durée de vie, etc.*
- **Attributs**
  - Couples (nom, valeur) utilisables par le système ou l'application pour sélectionner les messages
- **Données définies par l'application**
    - *Texte*
    - *Données structurées (XML)*
    - *Binaire*
    - *Objets (sérialisés)*
    - *. . .*

© ScalAgent Distributed Technologies – 2001-2003

Modes de désignation

- **Désignation indirecte**
  - Les entités communiquent via un objet intermédiaire : destination
    - *Destination : structure de données réceptacle de messages*
    - *Exemple : Queue (file) de messages*
- **Désignation de groupe**
  - groupe = ensemble de récipiendaires identifiés par un nom unique
    - *gestion dynamique du groupe : arrivée/départ de membres*
    - *différentes politiques de service dans le groupe : 1/N, N/N*
    - *Applications : répartition de charge (1/N), tolérance aux pannes (N/N)*
- **Désignation anonyme**
  - désignation associative : les destinataires d'un message sont identifiés par des propriétés (attributs du message)
  - Publish/Subscribe (Publication / Abonnement)

© ScalAgent Distributed Technologies – 2001-2003

Modes de communication

- 4 Relations entre producteur(s) et consommateur(s)
  - 1 producteur → 1 consommateur
  - 1 producteur → N consommateurs
  - P producteurs → 1 consommateur
  - P producteurs → N consommateurs

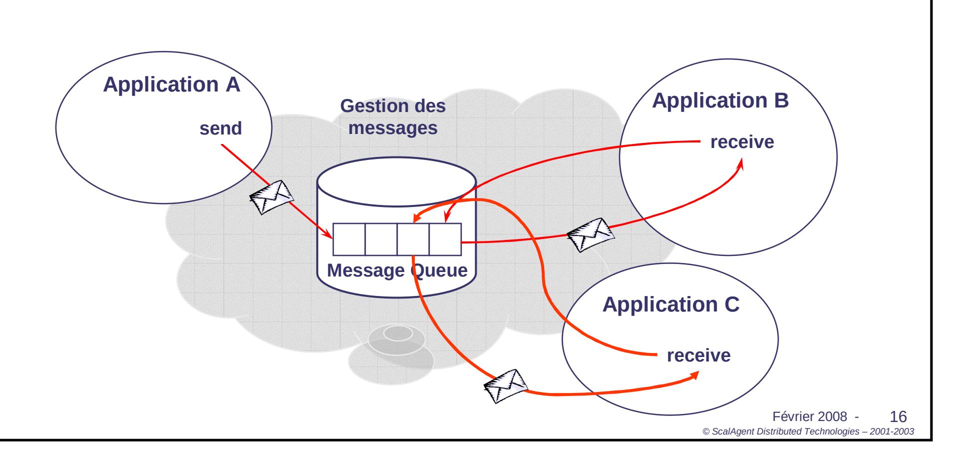
. . *mais seulement*

- 2 Modèles de communication de base
  - Point-To-Point : 1 producteur → 1 consommateur
  - Multi-points : 1 producteur → N consommateurs

© ScalAgent Distributed Technologies – 2001-2003

Modèle Point à point (1/2)

- Un message émis sur une queue de messages donnée est consommé par une unique application
  - asynchronisme et fiabilité



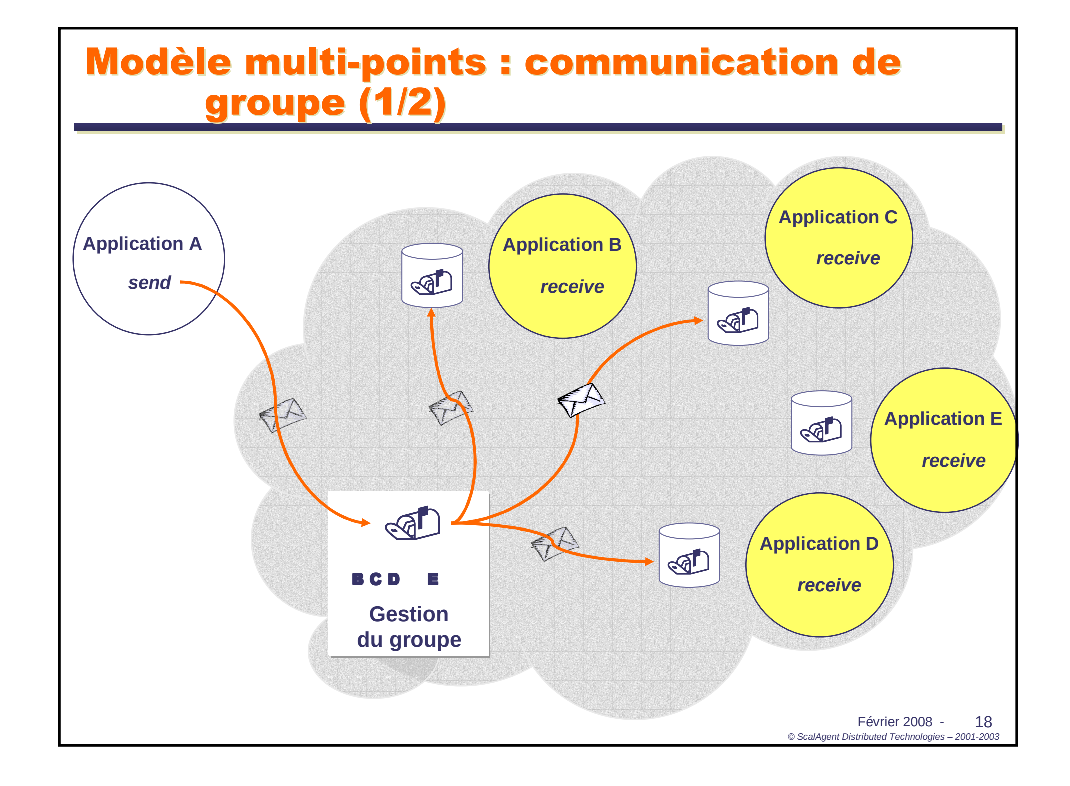
© ScalAgent Distributed Technologies – 2001-2003

Modèle Point à Point (2/2)

- **Séparation entre destination et consommateur**
  - destination statique : commune à plusieurs producteurs/consommateurs
  - Consommateur unique pour un message donné
- **Indépendance du producteur et du consommateur**
  - Via l'objet « Queue de messages »
  - Rend possible l'évolution de la relation producteur - consommateur
  - Indépendance temporelle : asynchronisme
- **Acquittement du traitement par le consommateur**
  - Acquittement de niveau système
    - *Libération de l'emplacement du message dans la queue*
    - *Fiabilisation de l'opération de consommation*
  - Acquittement de niveau applicatif : à programmer explicitement

© ScalAgent Distributed Technologies – 2001-2003

Modèle multi-points : communication de groupe (1/2)



© ScalAgent Distributed Technologies – 2001-2003

Communication de groupe (2/2)

- **Gestion du groupe**
  - Communication multi-points (1 producteur → N consommateurs)
  - Indépendance vis-à-vis des émetteurs
  - Gestion dynamique
    - *Arrivée / départ de membres*
- **Politiques de gestion des messages**
  - Persistance, durée de vie
  - Historique (forum)
- **Politiques de service des messages**
  - 1 / N : répartition de charge
  - N / N : réplication (tolérance aux pannes)
  - P / N : réplication partielle

© ScalAgent Distributed Technologies – 2001-2003

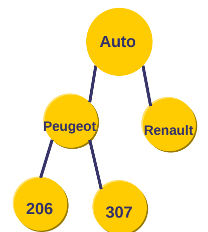
Modèle Publish/Subscribe (1/2)

- **Un message émis vers un sujet (*Topic)* donné est délivré à l'ensemble des applications abonnées à ce *Topic*.**

© ScalAgent Distributed Technologies – 2001-2003

Modèle Publish/Subscribe (2/2)

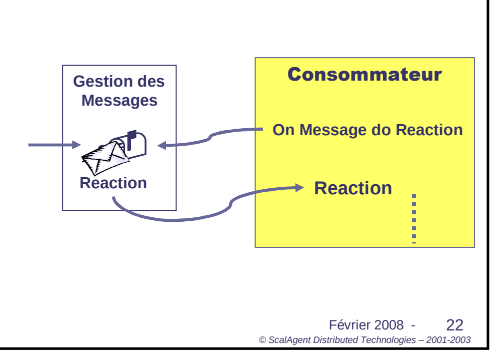
- **Relation producteur - consommateur**
  - Communication multi - points : 1 producteur → N consommateurs
  - Désignation anonyme via le Topic
  - Dépendance temporelle
    - *Le message est délivré aux consommateurs « actifs » lors de la production*
- **Critères d'abonnement**
  - Statique : « subject based »
    - *Organisation plate ou hiérarchique des sujets*
  - Dynamique : « content based »
    - *Implantation distribuée délicate*
- **Abonnements temporaires/durables**



© ScalAgent Distributed Technologies – 2001-2003

Modes de consommation

- **« *Pull* » – consommation explicite**
  - Les consommateurs programment explicitement l'accès aux messages
  - En cas d'absence de message : attente ou exception
- **« *Push* » – consommation implicite**
  - Une méthode prédéfinie (réaction) est attachée à la production d'un message (événement)
  - L'occurence d'un événement entraîne l'exécution de la réaction associée.
  - → *Modèle Evénement / Réaction*



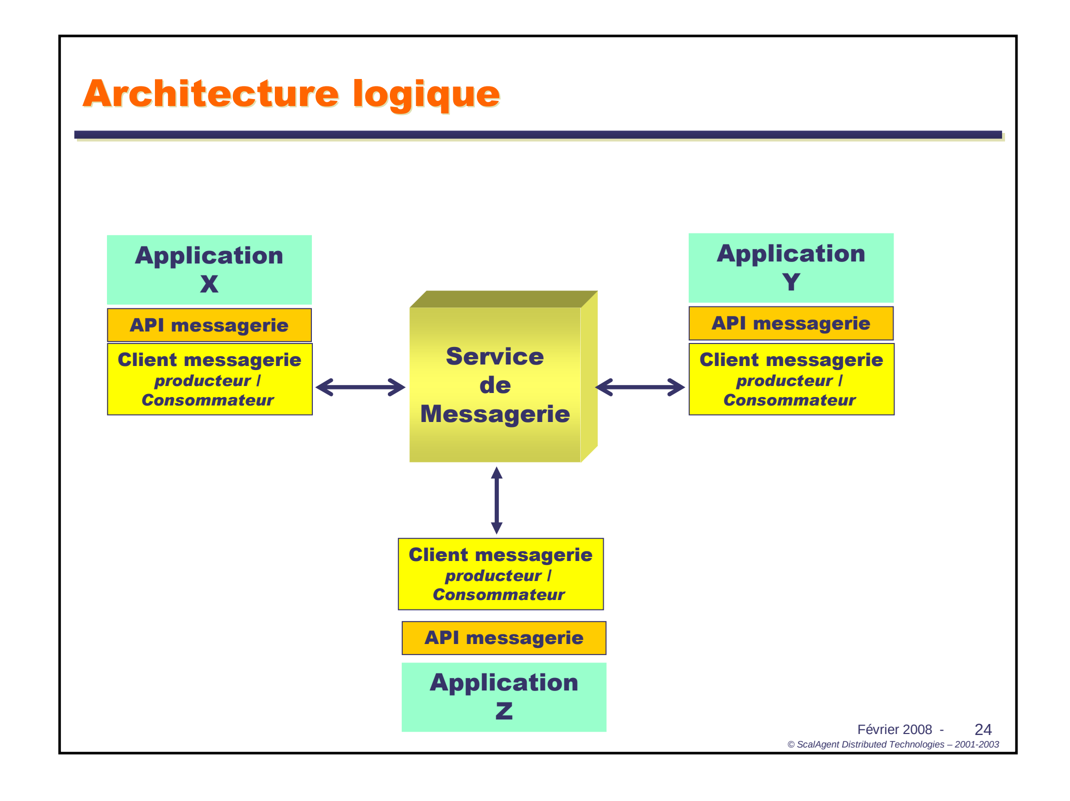
© ScalAgent Distributed Technologies – 2001-2003

Environnement d'exécution (run time)

- **Architecture logique**
  - Service de messagerie (*message server, message provider*)
    - *Gestion des messages (queues, topics)*
    - *Gestion des connexions avec les clients : producteurs, consommateurs*
  - Clients du service : producteurs, consommateurs
- **Architecture physique**
  - Centralisée, partitionnée, répartie
- **Qualité de service**
    - *Disponibilité, fiabilité*
    - *Sécurité*
    - *Performance*
    - *scalabilité*

© ScalAgent Distributed Technologies – 2001-2003

Architecture logique



© ScalAgent Distributed Technologies – 2001-2003

Architecture physique

- Serveur centralisée
  "Hub and Spoke"

Client Client Client Client Serv. MSG

- Serveur distribué
  "Snowflake"

Client Client Client Client Client Client Serv. MSG Serv. MSG Serv. MSG

- Serveur distribué
  Bus

Client Client Client Serv. MSG Serv. MSG Serv. MSG

© ScalAgent Distributed Technologies – 2001-2003

Qualité de service

- **Disponibilité et fiabilité**
  - Persistance des messages et Garantie de délivrance
    - *Au plus une fois, au moins une fois, exactement une fois*
- **Performance et scalabilité**
  - Nombre de sites, nombre de messages, taille des messages
    - *Réseau local (Intranet)*
    - *Réseaux hétérogènes à grande échelle (Internet)*
- **Transaction**
- **Sécurité**
- **Répartition de charge**
- **Ordonnancement**

© ScalAgent Distributed Technologies – 2001-2003

Administration

- Administration du MOM
  - Configuration, déploiement
  - Monitoring
- Administration du service de messagerie
  - Service de désignation
  - Service de messagerie : gestion des objets de communication : queues, topics, etc.
    - *Configuration, déploiement*
    - *Monitoring, reconfiguration*
  - Clients

© ScalAgent Distributed Technologies – 2001-2003

Présentation de JMS

- JMS en bref : objectifs et limitations
- Les concepts de JMS
- JMS en mode point à point
- JMS en mode Publish – Subscribe
- Qualité de service (QoS)

© ScalAgent Distributed Technologies – 2001-2003

JMS en bref (1/2)

- **Spécification d'un système de messagerie Java**
  - API Java entre une application et un bus à messages
  - Modèles de communication
    - *Point à point : message queuing*
    - *Multi points : publish – subscribe*
  - Divers types de données échangées : binaire, objets, texte, XML, . .
- **JMS ne définit pas le mode de fonctionnement du bus**
- **Les applications JMS sont indépendantes d'un bus (e.g. objectif de portabilité)**
  - … mais l'interopérabilité entre des plates-formes JMS hétérogènes nécessite la définition d'une passerelle entre les bus à messages

Février 2008 - 29

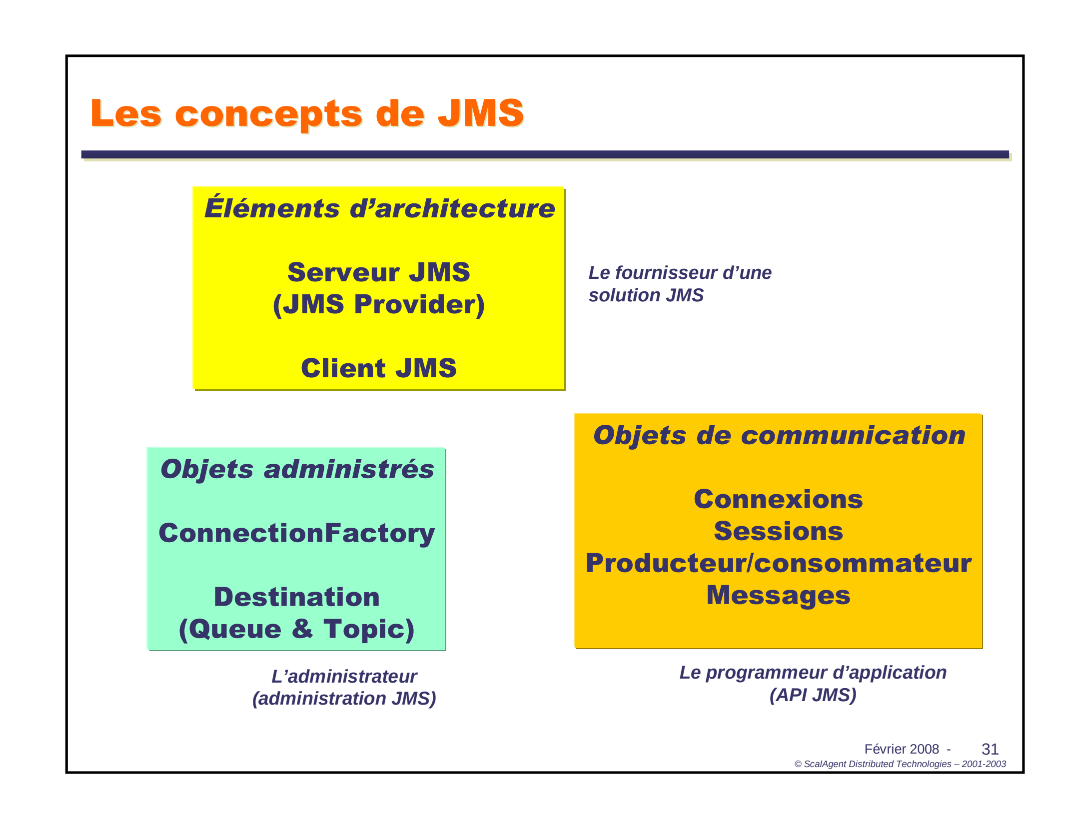
© ScalAgent Distributed Technologies – 2001-2003

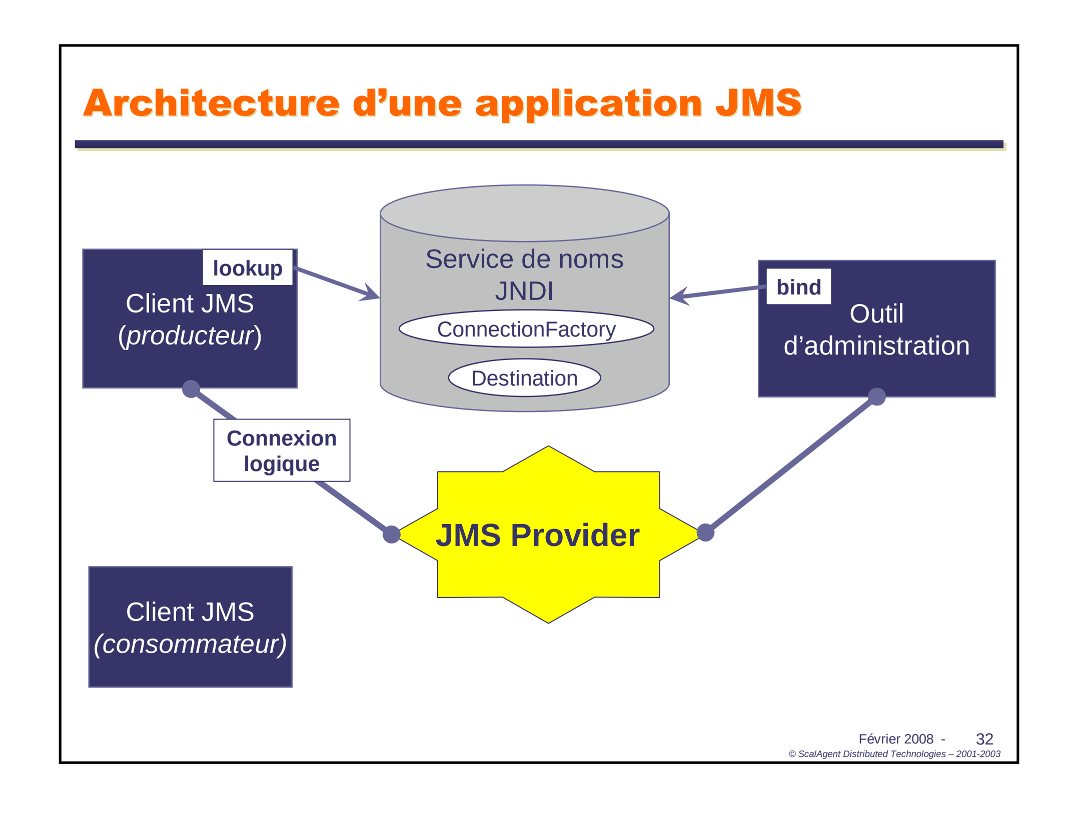
JMS en bref (2/2)

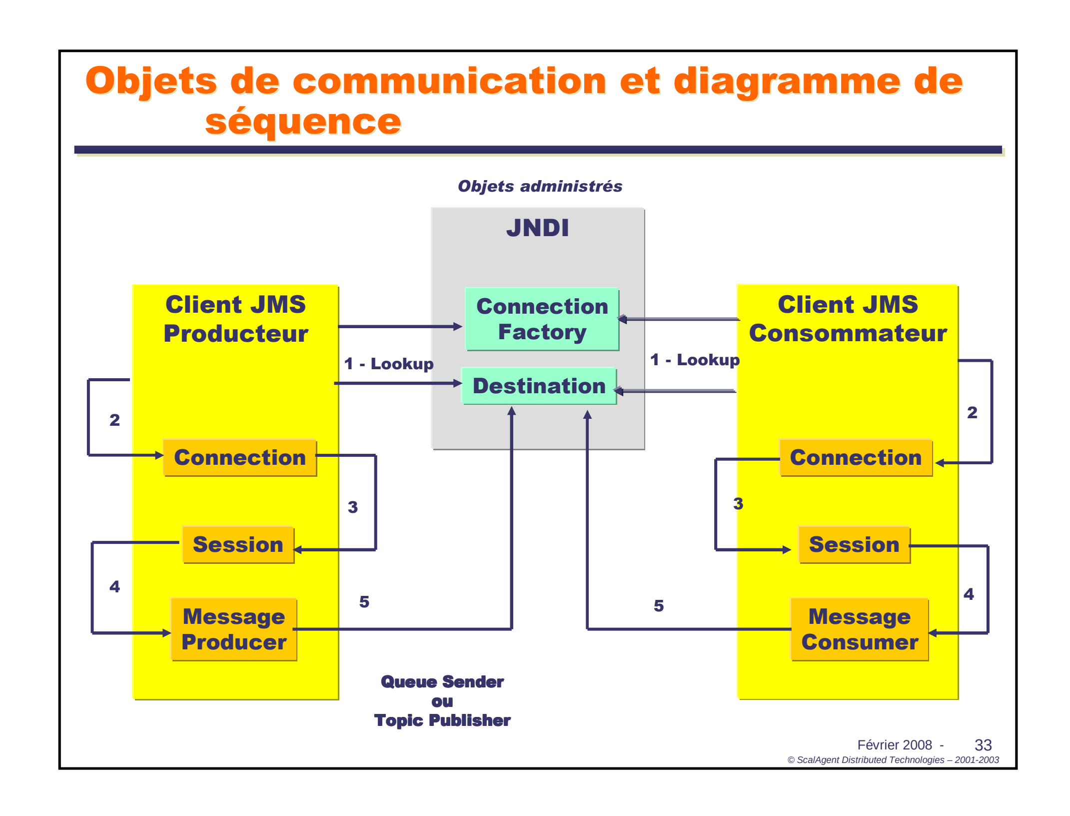
- **La spécification JMS n'est pas complète**
  - e.g. déploiement et administration sont spécifiques d'un produit donné
  - Les produits offrent des fonctions additionnelles : "topics" hiérarchiques, . .
- **Le support de JMS est un élément stratégique de la spécification J2EE**
- **La dernière spécification JMS 1.1. unifie la manipulation**
  - des "*queues*" : communication point-à-point
  - et des "*topics*" : communication multi-points (Publish-Subscribe)
    - *API simplifiée*
    - *Ressources réduites*

Février 2008 - 30

© ScalAgent Distributed Technologies – 2001-2003


Les concepts de JMS
Éléments d'architecture
Serveur JMS
(JMS Provider)
Client JMS
Le fournisseur d'une
solution JMS
Objets administrés
ConnectionFactory
Destination
(Queue & Topic)
Objets de communication
Connexions
Sessions
Producteur/consommateur
Messages
L'administrateur
(administration JMS)
Le programmeur d'application
(API JMS)
Février 2008 - 31
© ScalAgent Distributed Technologies – 2001-2003


Architecture d'une application JMS
lookup
Client JMS
(producteur)
Service de noms
JNDI
ConnectionFactory
Destination
bind
Outil
d'administration
Connexion
logique
JMS Provider
Client JMS
(consommateur)
Février 2008 - 32
© ScalAgent Distributed Technologies – 2001-2003



« Messaging Domains »

- **Point-to-Point**
- **Publish/Subscribe**
- **JMS 1.1 : unification des domaines**
  - Réduit et simplifie l'API (à terme)
  - Permet l'utilisation de Queues et Topics dans une même session (transaction)

Interface « parent »	*Point-à-point*	*Publish/Subscribe*
Destination	Queue	Topic
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver	TopicSubscriber

Février 2008 - 34
© ScalAgent Distributed Technologies – 2001-2003

Les objets JMS

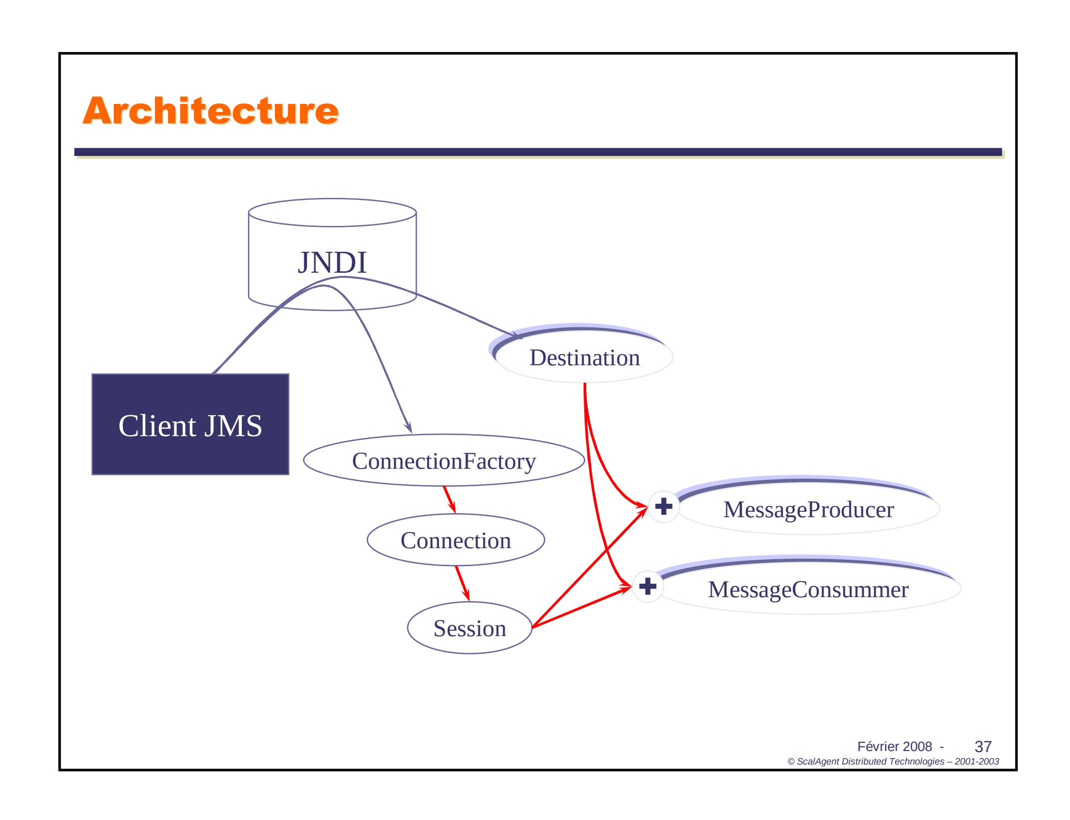
- **Objets administrés**
  - *ConnectionFactory* : point d'accès à un serveur MOM
  - *Destination* : *Queue* ou *Topic*
- **Objet *Connection***
  - Crée à partir d'un objet *ConnectionFactory*
  - Authentifie le client et encapsule la liaison avec le JMS provider
  - Gère les *Sessions*
- **Objet *Session***
  - Créé à partir d'un objet *Connection*
  - Fournit un contexte (transactionnel) mono-threadé de production/consommation de messages
  - Gère les acquittements de messages et les transactions , sérialise l'exécution des MessageListener,

© ScalAgent Distributed Technologies – 2001-2003

Les objets JMS

- **MessageProducer**
  - Fabriqué par la session → *QueueSender*, *TopicPublisher*
  - Permet l'émission de message → méthodes *Send*, *Publish*
- **MessageConsumer**
  - Fabriqué par la session → *QueueReceiver*, *TopicSubscriber*
  - Permet la réception de message
    - Synchrone → méthodes *Receive { (), (timeout)} ReceiveNoWait ()*
    - Asynchrone → objet *MessageListener* à l'écoute des messages
      exécution de la méthode *onMessage* de l'objet
  - Permet le filtrage des messages
    - *Syntaxe proche d'une condition SQL appliquée sur les champs de l'en-tête et des propriétés*

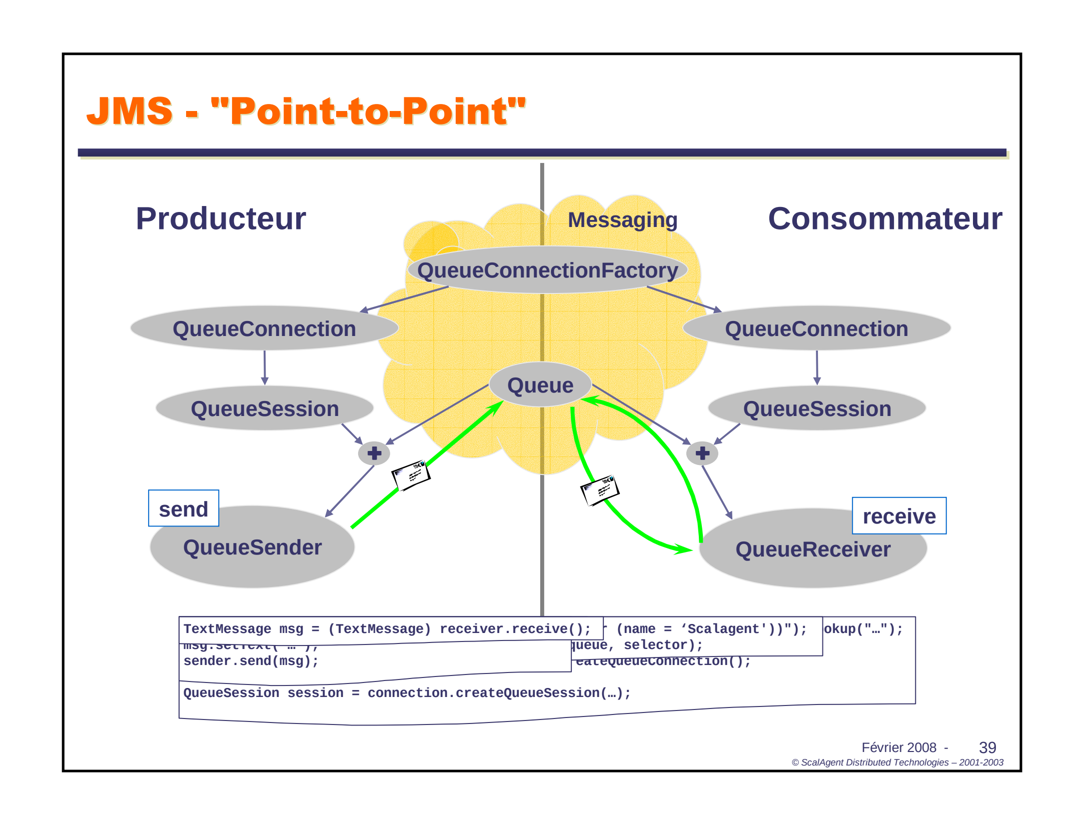
© ScalAgent Distributed Technologies – 2001-2003

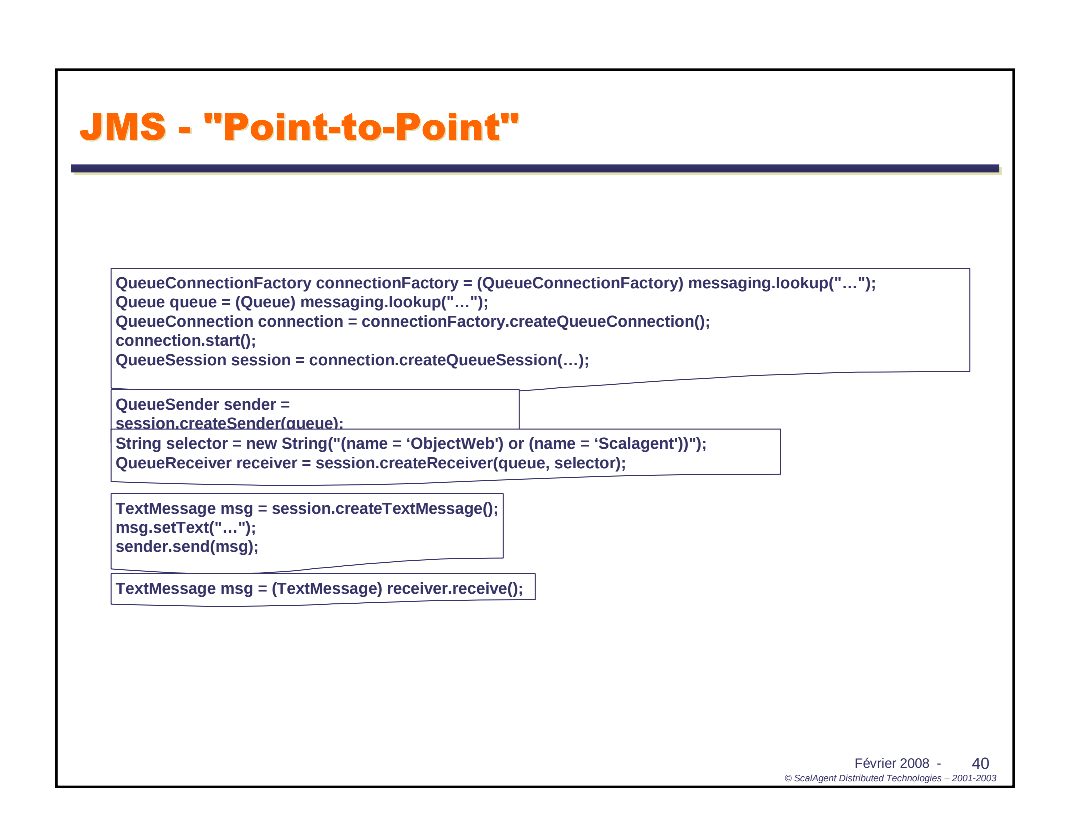


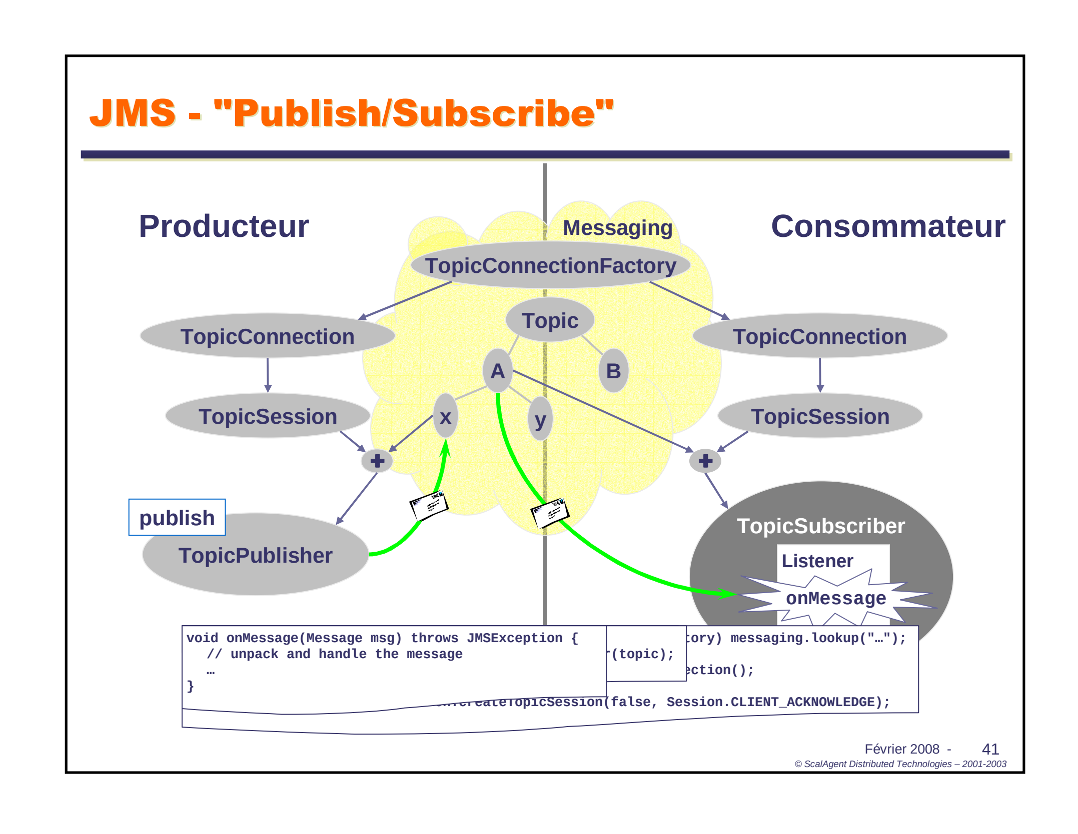
Le message JMS

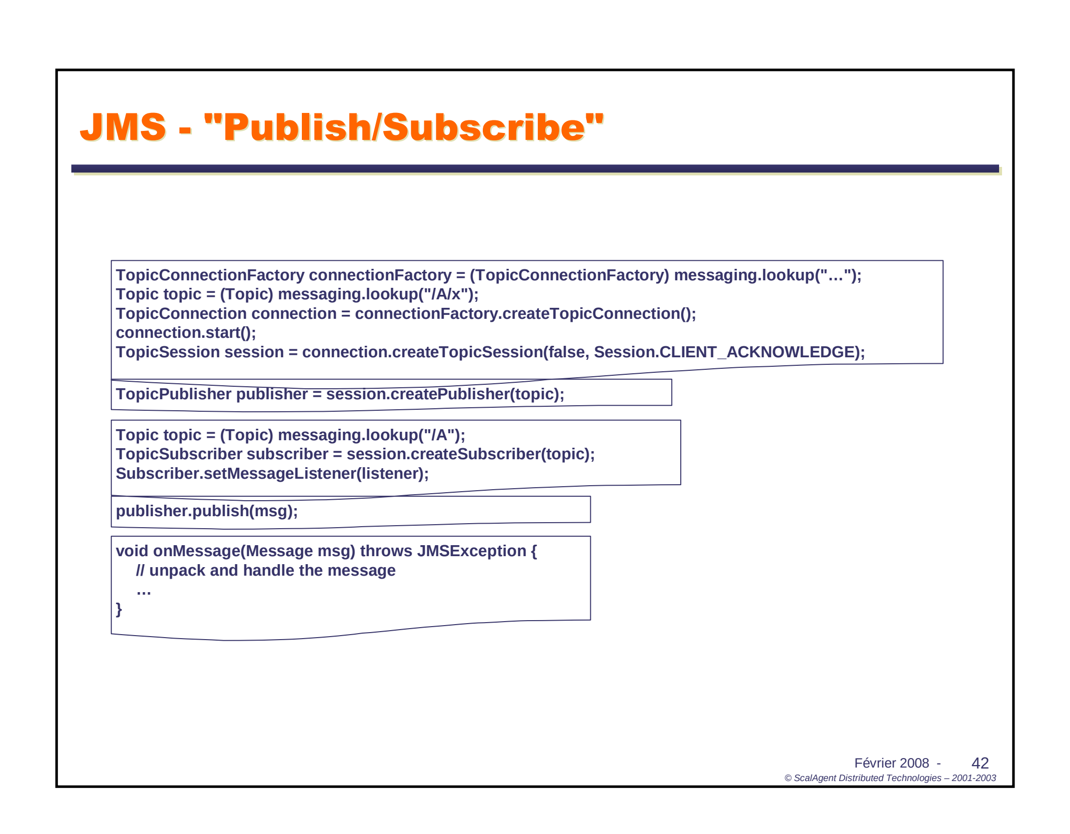
- **Entête**
  - JMSMessageId, JMSDestination, JMSDeliveryMode, JMSExpiration, JMSPriority, etc.
- **Propriétés**
  - Couple <nom, valeur>
- **Corps**
  - TextMessage, MapMessage
  - StreamMessage, ObjectMessage
  - BytesMessage

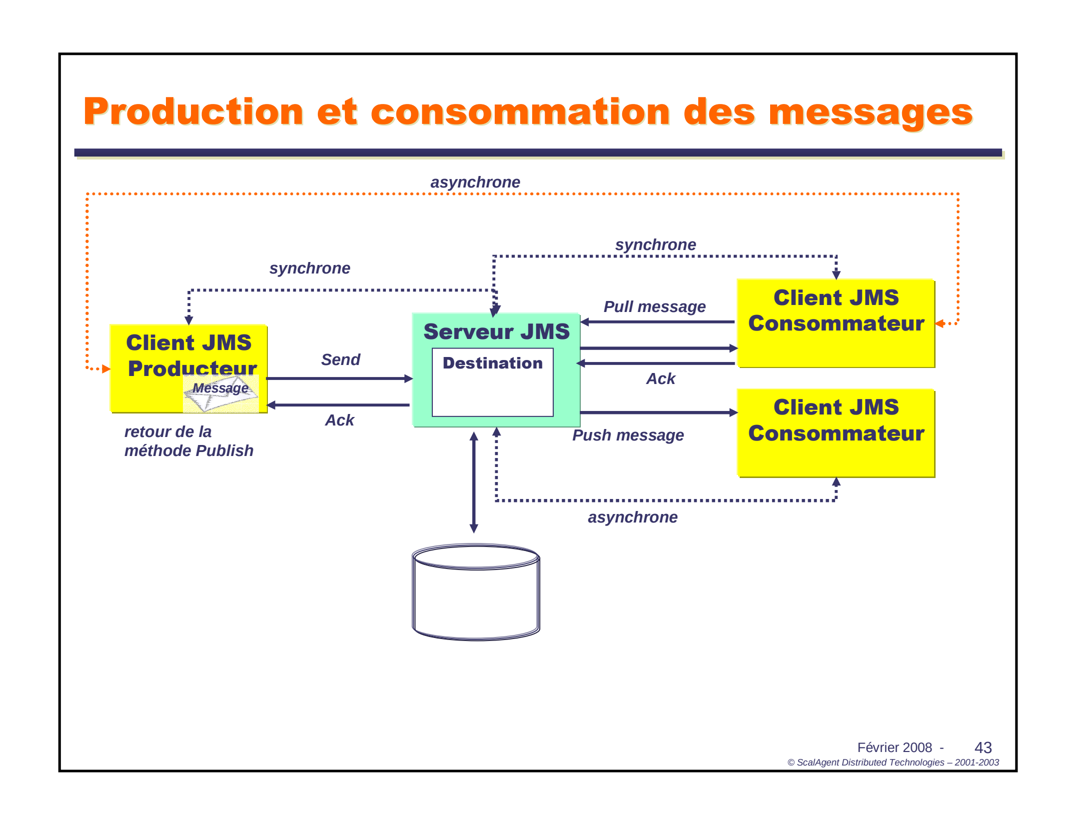
© ScalAgent Distributed Technologies – 2001-2003

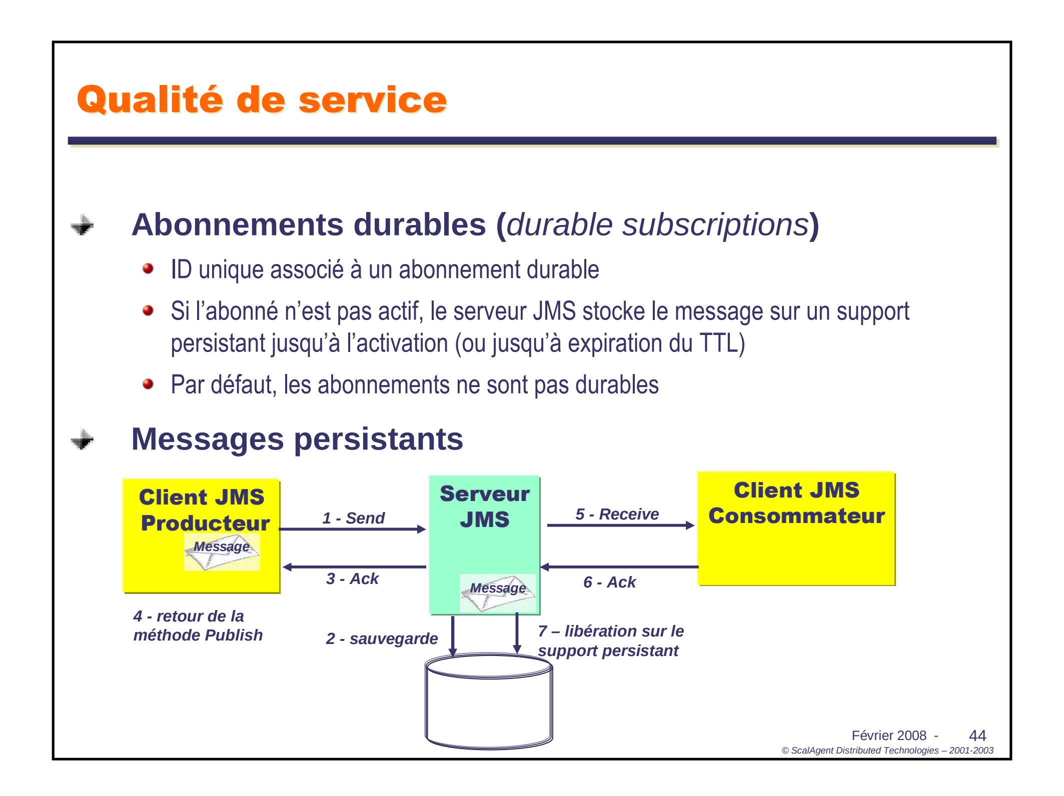

JMS - "Point-to-Point"
Producteur
Messaging
Consommateur
QueueConnectionFactory
QueueConnection
QueueConnection
QueueSession
QueueSession
Queue
send
QueueSender
receive
QueueReceiver
TextMessage msg = (TextMessage) receiver.receive();
sender.send(msg);
QueueSession session = connection.createQueueSession(…);
Février 2008 - 39
© ScalAgent Distributed Technologies – 2001-2003

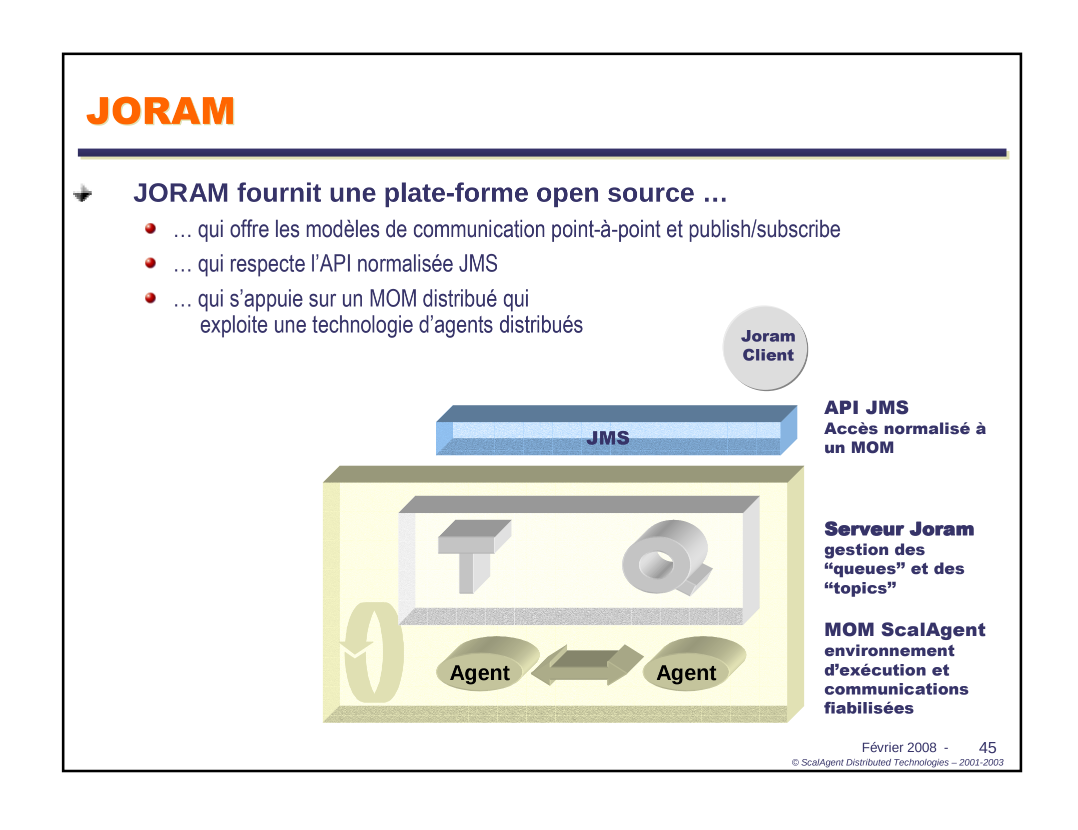

JMS - "Point-to-Point"
QueueConnectionFactory connectionFactory = (QueueConnectionFactory) messaging.lookup("…");
Queue queue = (Queue) messaging.lookup("…");
QueueConnection connection = connectionFactory.createQueueConnection();
connection.start();
QueueSession session = connection.createQueueSession(…);
QueueSender sender =
session.createSender(queue);
String selector = new String("(name = 'ObjectWeb') or (name = 'Scalagent')");
QueueReceiver receiver = session.createReceiver(queue, selector);
TextMessage msg = session.createTextMessage();
msg.setText("…");
sender.send(msg);
TextMessage msg = (TextMessage) receiver.receive();
Février 2008 - 40
© ScalAgent Distributed Technologies – 2001-2003

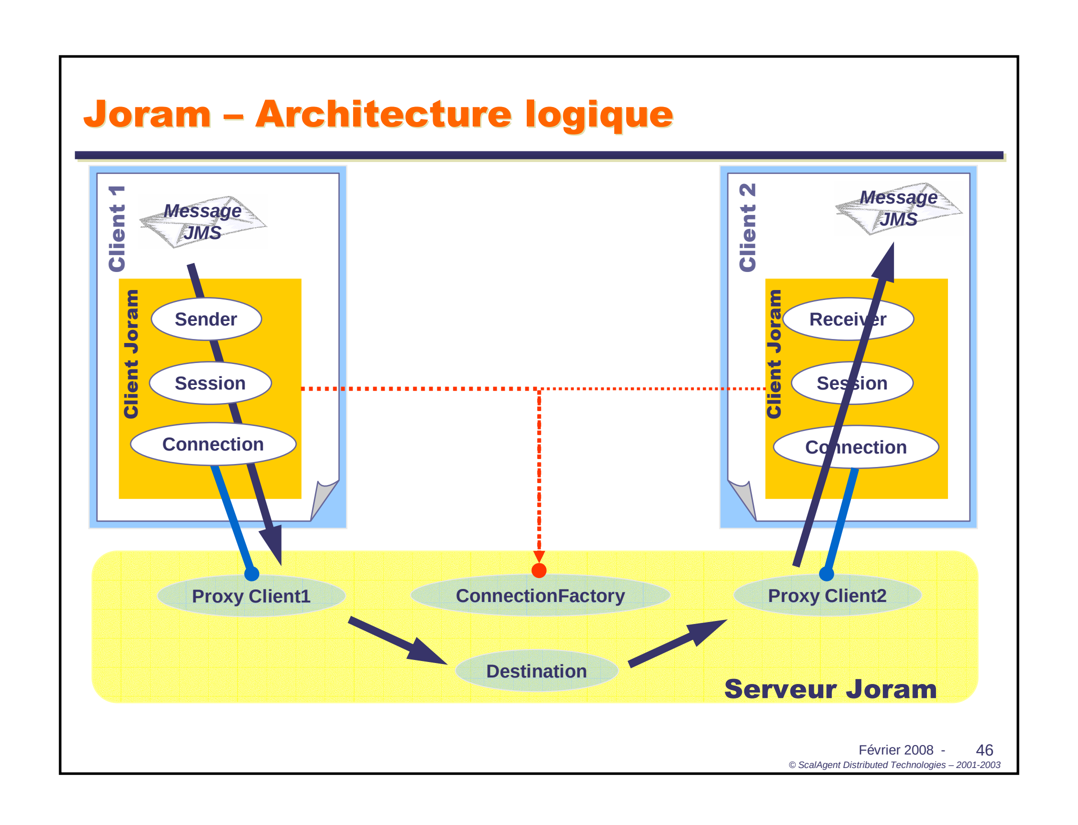

JMS - "Publish/Subscribe"
Producteur
Messaging
Consommateur
TopicConnectionFactory
TopicConnection
Topic
TopicConnection
A
B
TopicSession
x
y
TopicSession
publish
TopicPublisher
TopicSubscriber
Listener
onMessage
void onMessage(Message msg) throws JMSException {
// unpack and handle the message
…
}
Février 2008 - 41
© ScalAgent Distributed Technologies – 2001-2003

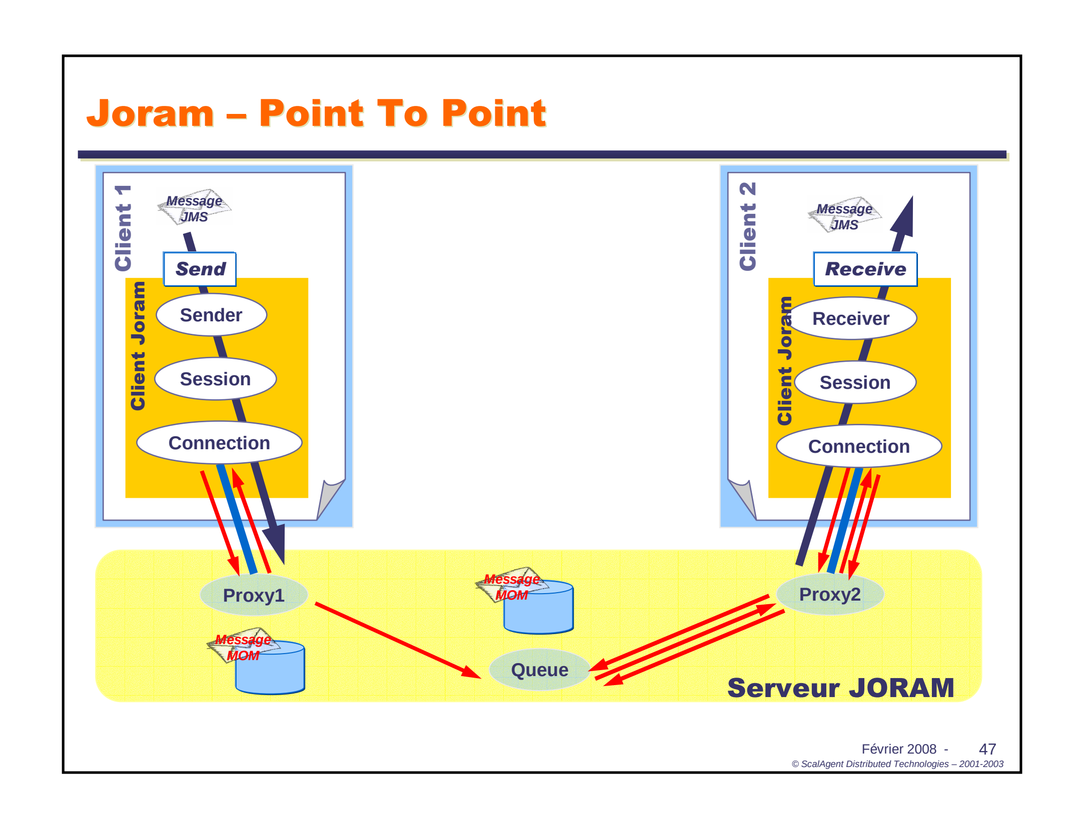

JMS - "Publish/Subscribe"
TopicConnectionFactory connectionFactory = (TopicConnectionFactory) messaging.lookup("…");
Topic topic = (Topic) messaging.lookup("/A/x");
TopicConnection connection = connectionFactory.createTopicConnection();
connection.start();
TopicSession session = connection.createTopicSession(false, Session.CLIENT_ACKNOWLEDGE);
TopicPublisher publisher = session.createPublisher(topic);
Topic topic = (Topic) messaging.lookup("/A");
TopicSubscriber subscriber = session.createSubscriber(topic);
Subscriber.setMessageListener(listener);
publisher.publish(msg);
void onMessage(Message msg) throws JMSException {
// unpack and handle the message
…
}
Février 2008 - 42
© ScalAgent Distributed Technologies – 2001-2003

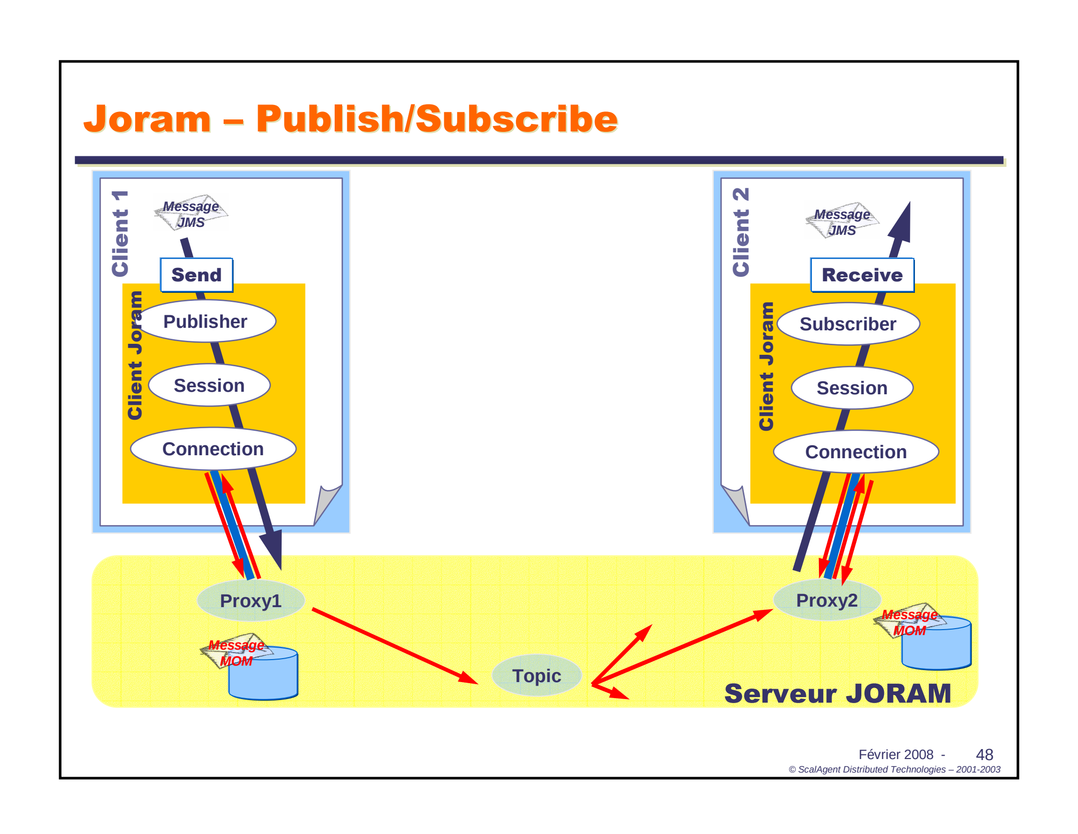

Production et consommation des messages
asynchrone
synchrone
synchrone
Pull message
Client JMS Consommateur
Client JMS Producteur
Message
Send
Serveur JMS
Destination
Ack
Ack
Client JMS Consommateur
Push message
retour de la méthode Publish
asynchrone
Février 2008 - 43
© ScalAgent Distributed Technologies – 2001-2003

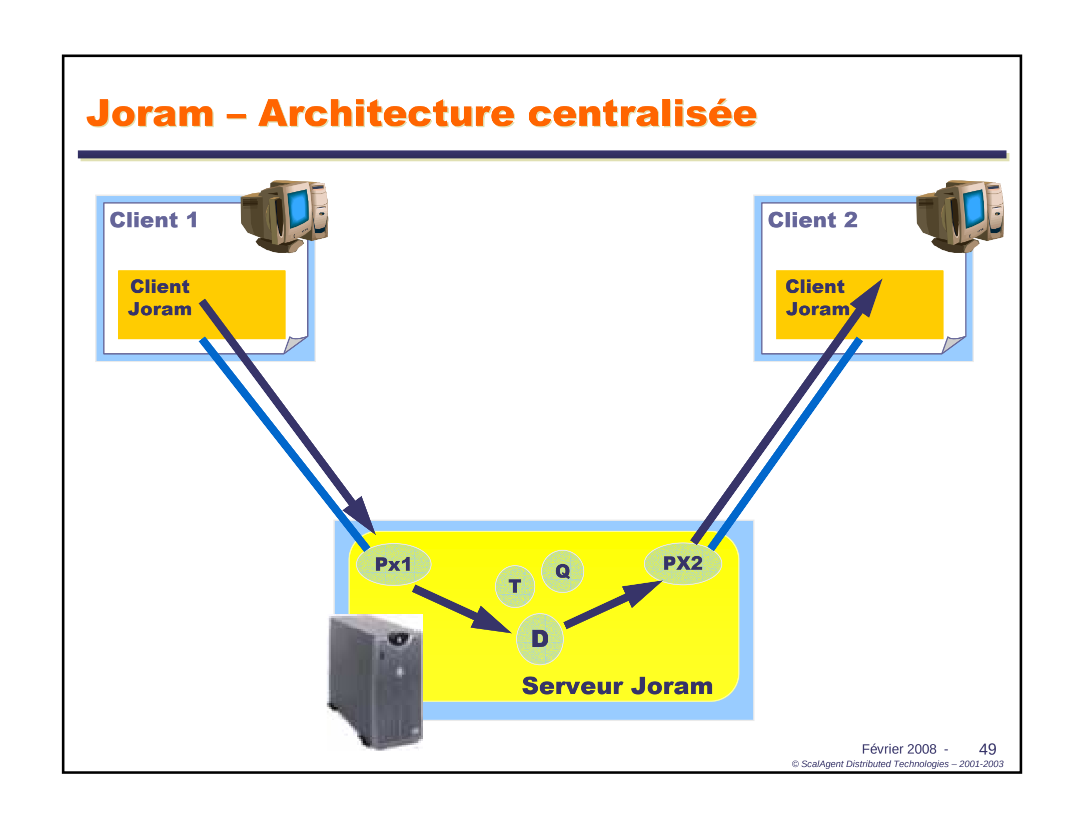

Qualité de service
Abonnements durables (*durable subscriptions*)
ID unique associé à un abonnement durable
Si l'abonné n'est pas actif, le serveur JMS stocke le message sur un support persistant jusqu'à l'activation (ou jusqu'à expiration du TTL)
Par défaut, les abonnements ne sont pas durables
Messages persistants
Client JMS Producteur
Message
1 - Send
Serveur JMS
5 - Receive
Client JMS Consommateur
3 - Ack
Message
6 - Ack
4 - retour de la méthode Publish
2 - sauvegarde
7 – libération sur le support persistant
Février 2008 - 44
© ScalAgent Distributed Technologies – 2001-2003

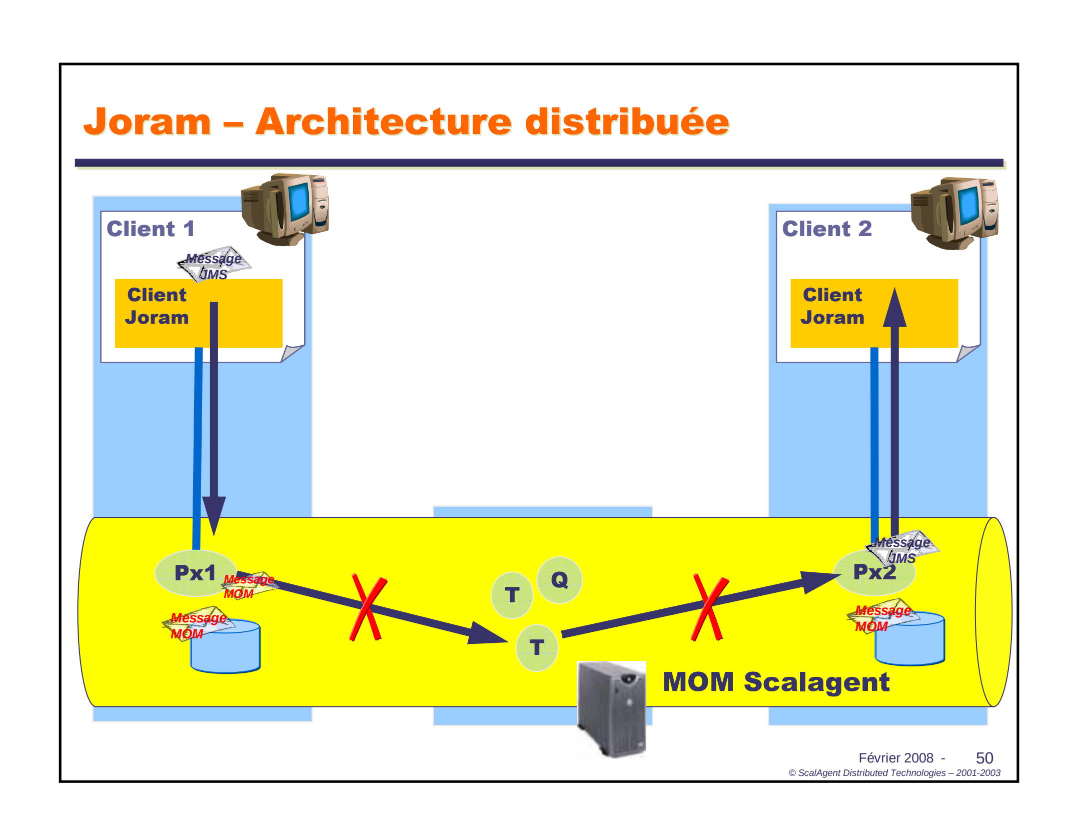

JORAM
JORAM fournit une plate-forme open source …
… qui offre les modèles de communication point-à-point et publish/subscribe
… qui respecte l'API normalisée JMS
… qui s'appuie sur un MOM distribué qui exploite une technologie d'agents distribués
Joram Client
JMS
API JMS
Accès normalisé à un MOM
Serveur Joram
gestion des "queues" et des "topics"
MOM ScalAgent
environnement d'exécution et communications fiabilisées
Agent
Agent
Février 2008 - 45
© ScalAgent Distributed Technologies – 2001-2003

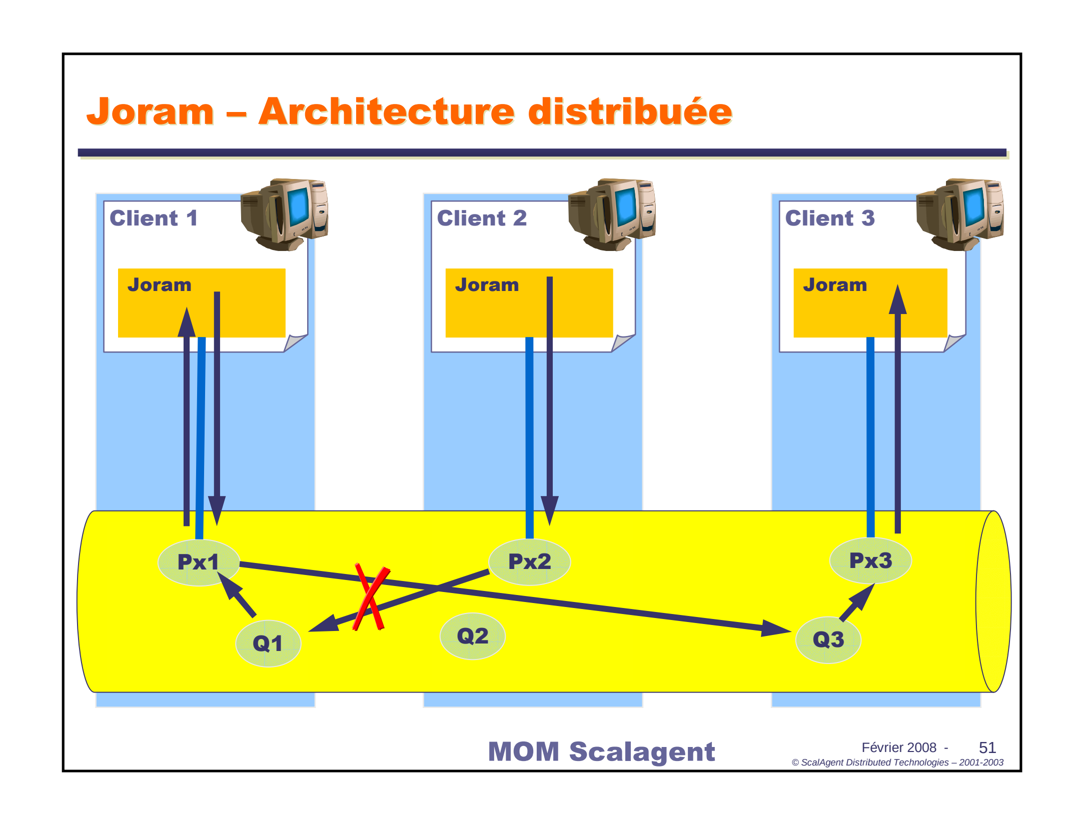

Joram – Architecture logique
Client 1
Message JMS
Client Joram
Sender
Session
Connection
Client 2
Message JMS
Client Joram
Receiver
Session
Connection
Proxy Client1
ConnectionFactory
Proxy Client2
Destination
Serveur Joram
Février 2008 - 46
© ScalAgent Distributed Technologies – 2001-2003

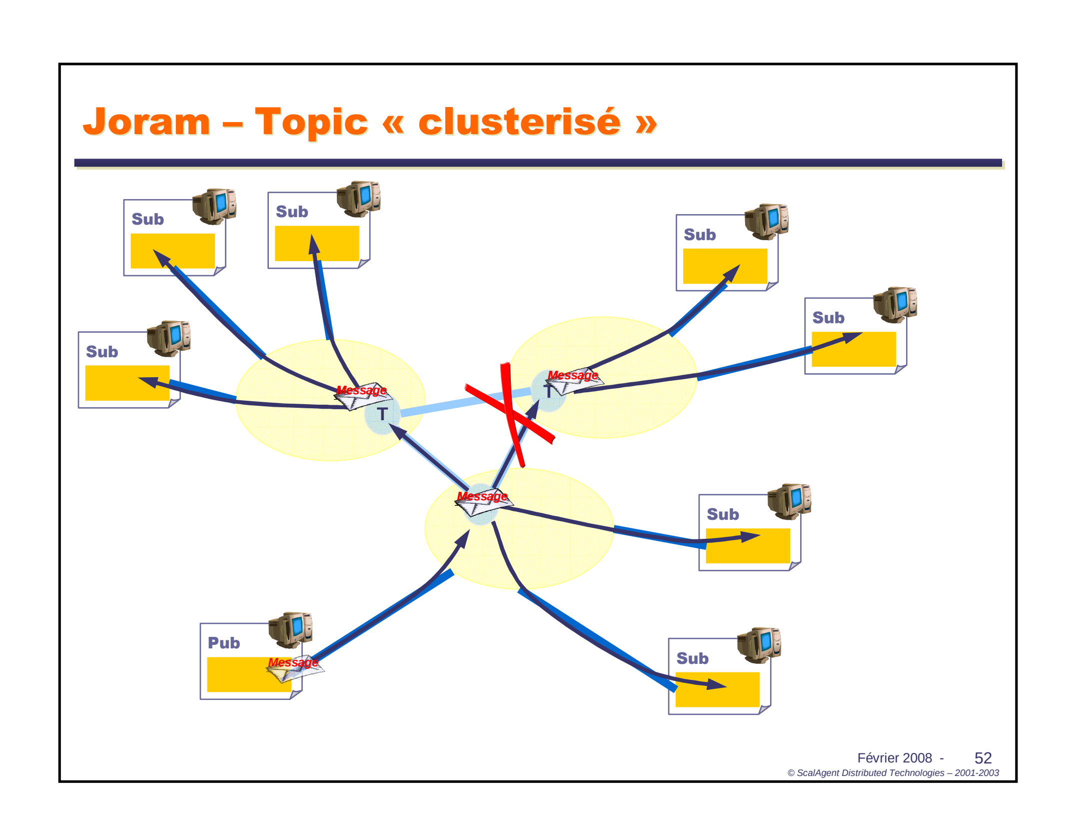

Joram – Point To Point
Client 1
Message JMS
Send
Client Joram
Sender
Session
Connection
Client 2
Message JMS
Receive
Client Joram
Receiver
Session
Connection
Proxy1
Message MOM
Message MOM
Queue
Proxy2
Serveur JORAM
Février 2008 - 47
© ScalAgent Distributed Technologies – 2001-2003

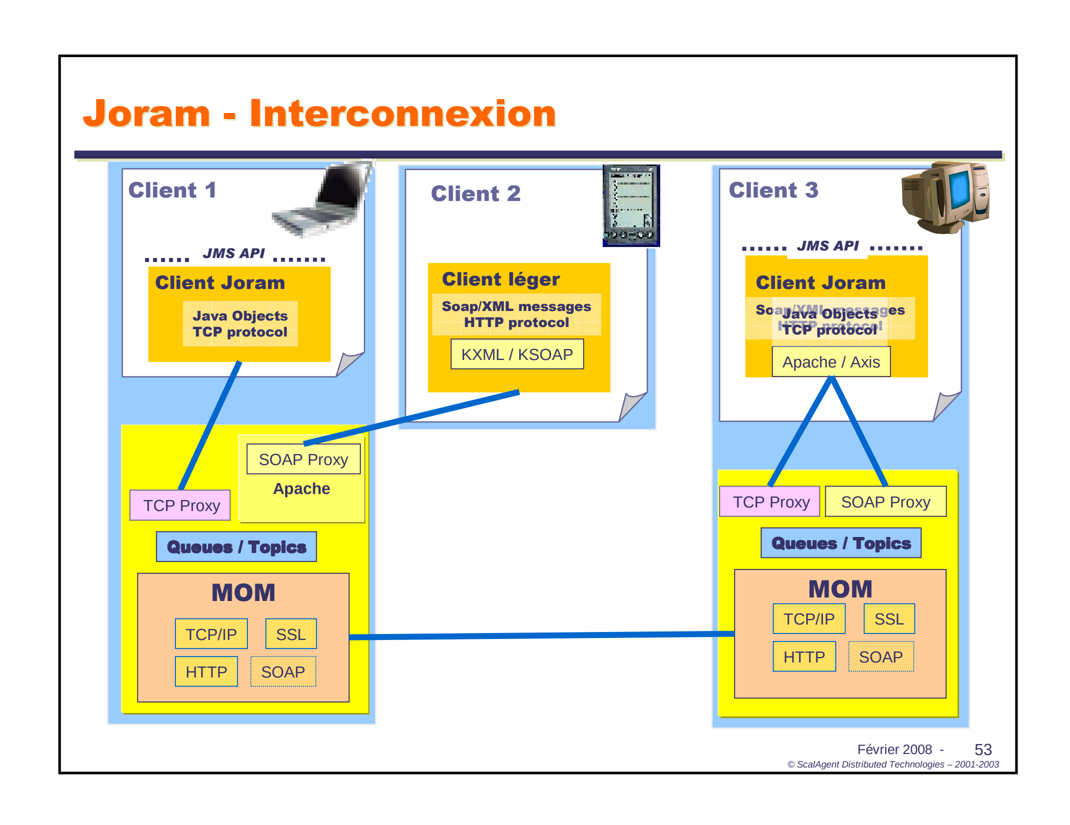

Joram – Publish/Subscribe
Client 1
Message JMS
Send
Client Joram
Publisher
Session
Connection
Client 2
Message JMS
Receive
Client Joram
Subscriber
Session
Connection
Proxy1
Message MOM
Topic
Proxy2
Message MOM
Serveur JORAM
Février 2008 - 48
© ScalAgent Distributed Technologies – 2001-2003


Joram – Architecture centralisée
Client 1
Client Joram
Client 2
Client Joram
Px1
T
Q
PX2
D
Serveur Joram
Février 2008 - 49
© ScalAgent Distributed Technologies – 2001-2003


Joram – Architecture distribuée
Client 1
Message JMS
Client Joram
Client 2
Client Joram
Px1
Message MOM
Message MOM
T
Q
T
Message JMS
Px2
Message MOM
MOM Scalagent
Février 2008 - 50
© ScalAgent Distributed Technologies – 2001-2003


Joram – Architecture distribuée
Client 1
Joram
Client 2
Joram
Client 3
Joram
Px1
Px2
Px3
Q1
Q2
Q3
MOM Scalagent
Février 2008 - 51
© ScalAgent Distributed Technologies – 2001-2003


Joram – Topic « clusterisé »
Sub
Sub
Sub
Sub
Sub
Sub
Sub
Pub
Message
Message
Message
Message
T
T
Février 2008 - 52
© ScalAgent Distributed Technologies – 2001-2003





- **Implantation open source de la spécification JMS**
  - API JMS : communication asynchrone entre applications Java
  - Bus à messages : médium de communication pour un environnement Internet (TCP/IP)
- **Disponible sur ObjectWeb**
  - http://joram.objectweb.org
  - Développé initialement par une équipe mixte Bull – Université – INRIA
  - Développement et support par *ScalAgent Distributed Technologies*
- **Usage double**
  - Service de messagerie autonome pour applications Java (J2EE à J2ME)
  - Composant de messagerie asynchrone intégré dans un serveur d'application J2EE (JonAS, Jboss, autres)

Février 2008 - 54

© ScalAgent Distributed Technologies – 2001-2003